METHOD AND APPARATUS FOR RECODING INSTRUCTIONS

Inventors: Soumya Banerjee
John L. Kelley
Ryan C. Kinter

FIELD OF THE INVENTION

[0001] The present invention relates generally to the field of computer architecture. More particularly, the present invention relates to the recoding of instructions.

BACKGROUND OF THE INVENTION

[0002] It is known that computer systems (e.g., main frames, personal computers, microprocessors, etc.) may be designed to execute instructions from one or more than one instruction set. In computer systems designed to execute instructions from more than one instruction set, for example, a first instruction set might be optimized for fast execution on a target system. However, instructions from this first set might have a relatively wide format (e.g., 32 or 64 bits in width) and therefore use a relatively large amount of memory space for storage. Hence, a second instruction set could be made available that is optimized for using less memory space through the use of a narrower instruction width format (e.g., 8 or 16 bits in width). Such instructions may execute routines slower than those from the first instruction set (because more and possibly different instructions are required to carry out the same function), but the narrower format contributes to a potential reduction in overall memory space required.

[0003] Additionally, a third instruction set could be made available to provide backwards compatibility to earlier generation machines that, again, may utilize instruction width formats of differing size (e.g., older 16-bit machines). Moreover, a fourth (or more) instruction set could be made available to provide upwards compatibility to new developments in instruction sets that may also

require different instruction width formats (e.g., 8-bit JAVA bytecodes). The foregoing examples, of course, are not exhaustive.

[0004] In order for a single computer system to support different instruction sets as described above, the system requires the capability to accommodate different instruction sets having potentially different instruction width formats. One way that such capability has been achieved in the past is by mapping one instruction set onto another, which allows a single decoder to be used for the different instruction width formats. Such mapping is possible, for example, where the one instruction set is a subset of the other. However, this is a significantly limiting feature because most instruction sets are not so related.

[0005] Moreover, this issue is made more complex in computer systems that simultaneously fetch a plurality of instructions for processing. Mapping may be achieved in such a system through a series of operations carried out in one or more pipeline stages (of a pipelined processor). These operations include reading a plurality of instructions from a cache memory, processing such instructions by comparing the tags of each instruction, selecting a desired instruction from the plurality (based on the tag compare) and then mapping the desired instruction. However, in such a serial mapping method, the processing of these instructions results in an increased branch penalty and/or cycle time.

[0006] Therefore, what is needed is a more efficient way of processing instructions for execution by a processor of a computer system.

BRIEF SUMMARY OF THE INVENTION

[0007] In one embodiment of the present invention, a computer architecture is provided for recoding. In embodiments, the architecture includes at least two interconnected recoders that are used to recode instructions. These recoders operate both independently and together when recoding instructions. As described herein, the present invention is embodied in various architectures, systems, apparatuses, computer program codes, and methods.

[0008] In embodiments of the present invention, the architecture is responsible, for example, for fetching instructions from an instruction cache, recoding instructions, and providing instructions to other pipe stages of a computer system. As described herein, in embodiments of the architecture, one or more instructions and cache tags are read from an instruction cache such as, for example, an on-chip memory block with multi-way associativity. The number of instructions and cache tags that are read from the instruction cache is dependent upon available bandwidth. After the instructions and cache tags are read, a tag compare and way selection unit checks the tags to verify that each desired instruction is available (i.e., present in the cache). An instruction staging unit stages and dispatches the fetched instructions to an instruction recoding unit. Because multiple instructions can be read from the instruction cache during a single clock cycle, the multiple instructions are staged and dispatched to the instruction recoding unit. The instruction recoding unit recodes the instructions received from the instruction staging unit to form recoded instructions that can be subsequently decoded and executed. In accordance with an embodiment of the present invention, the instruction recoding unit includes at least two interconnected recoders for recoding instructions. The recoded instructions produced by the instruction recoding unit are stored in an instruction buffer. This instruction buffer isolates the instruction fetch pipe stage operations of a computer system embodying the architecture from the operations of the other pipe stages of the computer system. In embodiments, an instruction bypass unit allows instructions to be passed directly from the tag compare and way selection unit to the instruction buffer.

[0009] Further embodiments, features, and advantages of the present invention are described in detail below with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS/FIGURES

[0010] The features and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference numbers indicate identical or functionally similar elements. Additionally, the left-most digit of a reference number identifies the drawing in which the reference number first appears.

[0011] FIG. 1A is a block diagram illustrating an example computer system.

[0012] FIG. 1B is a block diagram illustrating a series of pipeline or pipe stage operations carried out by the computer system of FIG. 1A.

[0013] FIG. 2 is a block diagram illustrating an example implementation of the instruction fetch pipe stage operations of FIG. 1B.

[0014] FIG. 3 is a schematic diagram of an example instruction fetch unit.

[0015] FIGs. 4A-B illustrate a flowchart of a method for performing the instruction fetch pipe stage operations of FIG. 1B, and which can be implemented by the instruction fetch unit of FIG. 3.

[0016] FIGs. 5A-B are diagrams that illustrate the recoding of an example expand instruction and an example expandable instruction.

[0017] FIGs. 6A-F are block diagrams that further illustrate the recoding of example instructions such as, for example, the expand and the expandable instructions of FIGs. 5A and 5B.

DETAILED DESCRIPTION OF THE INVENTION

[0018] FIG. 1A is a block diagram of an example computer system 100. Computer system 100 includes a pipelined processor 101, a memory 111, and a processor-memory bus 121. Processor 101 is coupled to processor-memory bus 121 via a cache controller 103 and a memory cache 107. Memory 111 is coupled to processor-memory bus 121 via a memory management unit (MMU) 113. A bus interface 133 couples an input/output (I/O) bus 131 to processor-memory bus

121. Three example input/output (I/O) controllers 135, 137, and 139 are shown coupled to I/O bus 131.

[0019] FIG. 1B is a block diagram illustrating a series of pipeline or pipe stage operations carried out by computer system 100. As illustrated in FIG. 1B, the pipe stage operations include instruction fetch operations 102, instruction decode and register fetch operations 104, execute and address calculation operations 106, memory access operations 108, and write back operations 110. The pipe stage operations identified in FIG. 1B are typical operations carried out, for example, by a reduced instruction set computer (RISC) architecture. In accordance with conventional RISC architecture, each pipe stage operation is allowed a single, uniform clock cycle to complete. Since the stages operate concurrently, the clock cycle is long enough to accommodate the slowest stage. Hence, once the pipeline of computer system 100 is full (i.e., each stage is processing one or more instructions), at least one instruction completes execution during each clock cycle. In addition to the stage operations of FIG. 1, alternate embodiments of system 100 may divide any single stage shown therein into multiple stages. For example, instruction fetch stage 102 may be divided into three stages that include instruction cache access in a first stage, tag comparison and way select in a second and instruction recoding in a third. Such alternate embodiments represent design choices that are well-known by those having ordinary skill in the art.

[0020] A detailed description of the conventional features of computer system 100 and its conventional pipe stage operations that would be known to persons skilled in the relevant computer arts is available, for example, in John L. Hennessy's and David A. Patterson's COMPUTER ARCHITECTURE: A QUANTITATIVE APPROACH, Third Edition (2003), available from Morgan Kaufman Publishers, San Francisco California, which is incorporated herein by reference in its entirety. Thus, these conventional features will not be further described herein. The description that follows focuses on novel and unconventional features of computer system 100 and its pipe stage operations that are not known to persons skilled in the relevant computer arts.

[0021] FIG. 2 is a block diagram illustrating an example implementation of the instruction fetch pipe stage operations 102. As illustrated in FIG. 2, the instruction fetch pipe stage operations 102 can be implemented with an instruction cache 202, a tag compare and way selection unit 204, an instruction staging unit 206, an instruction recoding unit 208, and an instruction bypass unit 210. The instruction fetch pipe stage operations 102 shown in FIG. 2 are responsible for fetching instructions and providing them to the other pipe stages of computer system 100. The instruction fetch pipe stage operations 102 shown in FIG. 2 are also responsible for handling the results of all control transfer instructions such as, for example, branch instructions and jump instructions.

[0022] In an embodiment, the instruction fetch pipe stage of computer system 100 operates as follows. First, one or more instructions and cache tags are read from instruction cache 202. Instruction cache 202 is a part of memory cache 107 and is preferably an on-chip memory block with multi-way associativity. The number of instructions and cache tags that are read from instruction cache 202 is dependent upon available bandwidth. For example, in one embodiment, sixty-four instruction data bits and a cache tag are read from instruction cache 202 in one clock cycle of computer system 100. This equates to eight 8-bit instructions, four 16-bit instructions, two 32-bit instructions, or one 64-bit instruction. A large bandwidth permits additional instructions and cache tags to be read in one clock cycle of computer system 100.

[0023] After the instructions and cache tag(s) are read, tag compare and way selection unit 204 checks the tag(s) to verify, for example, that each read instruction is the correct (i.e., desired) instruction. Other tag checks that can be performed include, for example, a lock check and a parity check.

[0024] Instruction staging unit 206 stages and dispatches instructions to instruction recoding unit 208. In an embodiment, as noted above, multiple instructions can be read from instruction cache 202 during a single clock cycle of computer system 100. Thus, when this occurs, the multiple instructions must be staged and dispatched to instruction recoding unit 208 if the number of fetched

instructions exceeds the parallel processing capabilities of instruction recoding unit 208.

[0025] Instruction recoding unit 208 recodes desired instructions received from instruction staging unit 206. The recoding operation of unit 208 maps instructions from one encoded state (e.g., a 16-bit instruction) to another encoded state (e.g., a 32-bit instruction). This is different from the decoding operation performed in pipestage operations 104 (Instruction Decode and Register Fetch) where an encoded instruction is decoded into one or more individual control signals for directing select operations within computer system 100. Instruction recoding unit 208 includes at least two interconnected parallel processing recoders for recoding instructions fetched from instruction cache 202. In an embodiment, instruction recoding unit 208 is capable of recoding instructions belonging to multiple instruction set architectures and instructions having different bit widths. How this is accomplished is further described below with regard to FIGs. 3-6. As illustrated in FIG. 3, the recoded instructions produced by instruction recoding unit 208 are stored in an instruction buffer 316. Instruction buffer 316 isolates the instruction fetch pipe stage operation of computer system 100 from the operations of the other pipe stages of computer system 100.

[0026] Instruction bypass unit 210 allows instructions to be passed directly from tag compare and way selection unit 204 to instruction buffer 316. In an embodiment, instruction bypass unit 210 is a data communications path. In other embodiments, instruction bypass unit 210 can include devices for partial or early decoding of instructions. Instruction bypass unit 210 is useful, for example, for allowing instructions that do not require recoding to be quickly placed in instruction buffer 316 or forwarded to the instruction decode pipe stage of computer system 100. In embodiments, processor 101 is equipped to decode and execute 32-bit instructions. In one embodiment, when a 32-bit instruction is fetched from instruction cache 202, it can be passed directly to instruction buffer 316 without recoding. On the other hand, a 16-bit instruction fetched from instruction buffer 202 would require recoding, and thus any 16-bit instruction

fetched from instruction cache 202 would be dispatched through instruction recoding unit 208 and the recoded instruction produced from instruction recoding unit 208 would be placed in instruction buffer 316 for subsequent decoding and execution by processor 101. Other instances where instruction bypass unit 210 may be useful will become apparent to persons skilled in the relevant computer arts based on the description provided herein.

[0027] FIG. 3 is a schematic diagram of an example instruction fetch unit 300 that implements instruction fetch pipe stage operations 102. Instruction fetch unit 300 includes an instruction cache 202, multiplexers 302, 304, 314a, and 314b, instruction bypass paths 210a and 210b, data flip-flops 306a, 306b, 306c, and 306d, recoders 310a and 310b, an information storage buffer 312, and an instruction buffer 316.

[0028] Instruction cache 202 is coupled to multiplexer 302. In an embodiment, this coupling provides a bandwidth of 64 data bits plus an associated tag (i.e., each 64 bits of data is associated with one tag). This bandwidth allows, for example, four 16-bit instructions or two 32-bit instructions to be read from instruction cache 202 during each read cycle. In an embodiment, instructions and tags are read from instruction cache 202 every other clock cycle of computer system 100 unless instruction buffer 316 is full. If instruction buffer 316 is full, the fetching of additional instructions from instruction cache 202 can be temporarily halted until instruction buffer 316 is again able to accept data.

[0029] Multiplexer 302 is used to implement the features of tag compare and way selection unit 204 described herein. The output of multiplexer 302 is 64 data bits. These bits can be provided to instruction buffer 316 by way of instruction bypass paths 210a and 210b or provided to multiplexer 304, via data flip-flops 306a-d for instruction staging.

[0030] Multiplexer 304 and data flip-flops 306a-d are used to implement the instruction staging features described above with regard to instruction staging unit 206. Multiplexer 304 is coupled to at least two recoders 310. In an embodiment, the data associated with data flip-flops 306a and 306b are operated upon by

multiplexer 304 and dispatched to recoders 310a and 310b, respectively during one clock cycle of computer system 100 (i.e., the data associated with data flip-flops 306a is dispatched to recoder 310a and the data associated with data flip-flops 306b is dispatched to recoder 310b). In the next clock cycle of computer system 100, the data associated with data flip-flops 306c and 306d are operated upon by multiplexer 304 and dispatched to recoders 310a and 310b, respectively (i.e., the data associated with data flip-flops 306c is dispatched to recoder 310a and the data associated with data flip-flops 306d is dispatched to recoder 310b). This processing permits proper staging of multiple instructions in embodiments where more instructions are fetched during an instruction cache read than can be processed in parallel by the available recoders 310. As noted above, in an embodiment such as the one depicted in FIG. 3, instruction cache 202 is read every other clock cycle of computer system 100 while recoding of the fetch instructions is being performed.

[0031] As shown in FIG. 3, recoders 310a and 310b are coupled to each other and to multiplexers 304, 314a, and 314b. Recoders 310a and 310b can be configured and operated so as to recode any given instruction or set of instructions to any particular desired instruction. For example, if it is desired that computer system 100 execute two different instruction set architectures, one having X-bit width instructions and one having Y-bit width instructions, Y being greater than X, then recoders 310a and 310b can be configured to recode X-bit width instructions, for example, to form Y-bit width recoded instructions or configured to recode both X-bit width instructions and Y-bit width instructions to form Z-bit width instructions. Recoders 310a and 310b can also be configured, for example, to recode instructions belonging to one instruction set to instructions of another instruction set, thereby facilitation program code portability, program code backwards compatibility and/or program code forwards compatibility. As will be appreciated by persons skilled in the relevant arts given the description herein, the possible configurations of recoders 310a and 310b and their ability to recode various instructions are potentially limitless.

[0032] The coupling of each recoder 310 to multiplexer 304 permits parallel recoding of instructions dispatched by multiplexer 304 to recoders 310a and 310b. Parallel recoding, in conjunction with the storing of recoded instructions in instruction buffer 316, decouples the instruction fetch pipe stage operations of computer system 100 from other pipe stage operations of computer system 100 and permits instruction fetch unit 300 to get ahead of, for example, instruction decoding and execution operations. By getting ahead, instruction fetch unit 300 shields the other pipe stage operations of computer system 100 from instruction fetch penalties such as cache misses and improves the overall operating performance of computer system 100.

[0033] The coupling as shown in FIG. 3, for example, of an output of recoder 310a to an input of recoder 310b and the coupling of an output of recoder 310b to an input of recoder 310a, via information storage buffer 312, permits recoders 310a and 310b to operate together to recode expand and expandable instructions (interrelated instructions that are illustrated in FIGs. 5A and 5B, and described in detail below). This joint recoding of expand and expandable instruction by recoders 310a and 310b avoids the insertion of instruction gaps or bubbles into instruction buffer 312 as a result of recoding delays that might otherwise be unavoidable when recoding these types of instructions.

[0034] Multiplexers 314a and 314b select which data bits are provided to instruction buffer 316. Each multiplexer 314a and 314b is coupled to an output of a recoder 310 and an instruction bypass path 210. In an embodiment, an operating mode of computer system 100, represented by one or more mode bits, is used to control multiplexers 314a and 314b thereby selecting when recoders 310 are bypassed.

[0035] Instruction buffer 316 is a conventional first-in first-out (FIFO) buffer. As noted above, buffer 316 helps to decouple the instruction fetch pipe stage operations of computer system 100 from other pipe stage operations of computer system 100 and permits instruction fetch unit 300 to get ahead of, for example,

instruction decoding and execution operations. In an embodiment, when instruction buffer 316 is full, cache reads are temporarily halted.

[0036] As shown in the embodiment of FIG. 3, instruction recoding unit 208 includes two recoders; i.e., 310a and 310b. However, alternative embodiments of the present invention may include more than two recoders (operating in parallel, series or both). As would be apparent to one of ordinary skill in the art, the construction and operation of such alternative embodiments would be similar to and logical extensions of the two recoder embodiment described herein. Additionally, the embodiment of FIG. 3 shows a recoding operation that receives 16-bit instructions and produces 32-bit instructions. In alternate embodiments, the recoding operation may receive and produce instructions of different sizes than shown herein. Additionally, such operations may concurrently accommodate multiple-sized instructions. For example, 16-bit and 32-bit instructions may both be recoded to a different sized instruction altogether (e.g., 35 bits) to accommodate the unique characteristics of each instruction set.

[0037] FIGs. 4A and 4B illustrate a flowchart of a method 400 for performing instruction fetch pipe stage operations 102. Method 400 can be implemented, for example, by instruction fetch unit 300.

[0038] Method 400 starts at step 402. In step 402, a plurality of instructions are fetched (read) from an instruction cache. Preferably, the number of instructions fetched in step 402 will be equal to or greater than the number of recoders available to recode fetched instructions.

[0039] In step 404, instructions fetched in step 402 are dispatched to each recoder available for recoding an instruction.

[0040] In step 406, a determination is made as to whether an instruction to be recoded is a desired instruction. If the instruction to be recoded is a desired instruction, control passes to step 408. If the instruction to be recoded is not a desired instruction, control passes to step 420.

[0041] In step 408, it is noted that steps 410 through 418 of method 400 are performed for each recoder available to recode an instruction fetched form an instruction cache in step 402.

[0042] In step 410, a determination is made by each available recoder as to whether the instruction to be recoded is an expand instruction. This determination can be made, for example, by examining the instruction's opcode. If the instruction to be recoded is an expand instruction, control passes to step 412. If the instruction to be recoded is not an expand instruction, control passes to step 416.

[0043] An example of an expand instruction is provided in FIGs. 5A and 5B. As used herein, an expand instruction is an instruction having data bits that are added or concatenated to bits of a second expandable instruction, thereby expanding an immediate value held in an immediate field of the expandable instruction. An example of an expandable instruction is also provided in FIG. 5A or FIG. 5B. Each expand instruction has an associated expandable instruction with which it must be paired during recoding or else the expanded immediate value formed during recoding, by combining data bits of the expand instruction and bits of an immediate field of the expandable instruction will result in an incorrectly recoded instruction. Persons skilled in the relevant computer arts will recognize expand and expandable instructions, as used herein, as being similar to MIPS16e(TM) instructions having similar functionality (e.g., the so-called "EXTEND" instruction). Additional information regarding the MIPS16e(TM) architecture may be found in the following publication, which is available from MIPS Technologies, Inc., and hereby incorporated by reference in its entirety: MIPS32(TM) Architecture for Programmers Volume IV-a: The MIPS16e(TM) Application-Specific Extension to the MIPS32(TM) Architecture, Rev. 2.00, MIPS Technologies, Inc. (2003). The expand and expandable instructions described herein, however, are not limited to just the functionality available with MIPS16e(TM) instructions.

[0044] In step 412, information is obtained by a recoder regarding an expand instruction, which is needed to recode the expand instruction's associated expandable instruction. At a minimum, this information will include one or more data bits of the expand instruction that are to be added or concatenated to one or more data bits of the associated expandable instruction during recoding. The actual minimum amount of information needed to recode a given pair of expand and expandable instructions according to the present invention will be dependent upon the configuration of the recoders used to recode these instructions.

[0045] In step 414, the information obtained in step 412 is passed to the recoder that needs the information to recode the associated expandable instruction. In an embodiment, this information is passed with other information such as the fact that an expand instruction has been detected.

[0046] In step 416, a determination is made as to whether the instruction to be recoded is an expandable instruction. In an embodiment, this determination can be made, for example, by examining the instruction's opcode and/or information passed by another recoder. If the instruction to be recoded is an expandable instruction, control passes to step 418. If the instruction to be recoded is not an expandable instruction, control passes to step 419.

[0047] In step 418, an expandable instruction is recoded based on information passed by another recoder (e.g, in step 414). As noted herein, the recoding process used is dependent on the configuration and operation of the particular recoder used to recode the expandable instruction. FIGs. 5A and 5B illustrate the recoding process for expand and expandable instructions.

[0048] In step 419, a normal (e.g., a non-expand or non-expandable) instruction is recoded without any need for information passed by another recoder. Again, as noted herein, the recoding process used in step 419 is dependent on the configuration and operation of the particular recoder used to recode the normal instruction.

[0049] In step 420, a determination is made whether there are additional instructions, fetched in step 402, that need to be recoded. If there are additional

instructions that need to be recoded, control passes to step 404. If there are no additional instructions, fetched in step 402, that need to be recoded, control passes to step 422.

[0050] In step 422, a determination is made whether there are additional instructions to be fetched from the instruction cache. If there are additional instructions to be fetched, control passes to step 402. Otherwise, control passes to step 424.

[0051] In step 424, method 400 ends.

[0052] FIG. 5A is a diagram that illustrates the process of recoding of an example expand instruction 500 and an example expandable instruction 510 to form a recoded instruction 520.

[0053] Expand instruction 500 includes an opcode field 502 and an expand field 504. As shown in FIG. 5A, instruction 500 has a width of X-bits (B_x).

[0054] Expandable instruction 510 includes an opcode field 512 and an immediate field 514. Fields 512 and 514 are not the only fields of expandable instruction 510. Expandable instruction 510 can be any instruction having an immediate field such as, for example, a jump instruction, a branch instruction, a memory read instruction, a memory write instruction, et cetera. As shown in FIG. 5A, instruction 510 also has a width of X-bits (B_x).

[0055] Recode instruction 520 is formed by adding or concatenating the bits of expand field 504 and immediate field 514 to form an expanded immediate field in instruction 520. The opcode field 522 of recoded instruction 520 directs computer system 100 to perform the operation or operations indicated by opcode field 512 of instruction 510. As shown in FIG. 5A, in an embodiment, recoded instruction 520 has a width of Y-bits (B_y).

[0056] FIG. 5B is a diagram that illustrates the process of recoding a second example expandable instruction 511 to form a recoded instruction 530.

[0057] Expandable instruction 511 includes an opcode field 513 and an immediate field 515. In this case, fields 513 and 515 are the only fields of expandable instruction 511. Expandable instruction 511 is representative of

instructions having functionality similar to MIPS16e(TM) jump and link (JAL) instructions or jump and link and switch operating modes (JALX) instructions.

[0058] Recode instruction 530 is formed by adding or concatenating the bits of expand field 504 and immediate field 515 to form an expanded immediate field in instruction 530. The opcode field 532 of recoded instruction 530 directs computer system 100 to perform the operation or operations indicated by opcode field 513 of instruction 511. As shown in FIG. 5B, in an embodiment, recoded instruction 530 also has a width of Y-bits (B_y).

[0059] FIGs. 6A-F are block diagrams that further illustrate the operation of a two-recoder embodiment of computer system 100, i.e., the embodiment illustrated by instruction fetch unit 300, and the recoding of instructions such as, for example, the expand and the expandable instructions of FIGs. 5A and 5B.

[0060] FIG. 6A illustrates an example recoding operation wherein four normal (e.g., non-expand and non-expandable) instructions I_0, I_1, I_2, and I_3 are fetched from an instruction cache during a clock cycle-0 of computer system 100. Instruction I_0 is dispatched to a recoder 310a. Instruction I_1 is dispatched to a recoder 310b. Because both of the instructions are normal instruction, recoders 310a and 310b are able to operate independently and recode both instructions I_0 and I_1 during one clock cycle of computer system 100. In the next clock cycle of computer system 100, instruction I_2 is dispatched to recoder 310a and instruction I_3 is dispatched to recoder 310b. Again, because both of these instructions are normal instructions, recoders 310a and 310b are able to operate independently and recode instruction I_2 and I_3 during a single clock cycle of computer system 100. Thus, at the end of two clock cycles of computer system 100, all four instructions I_0, I_1, I_2, and I_3 have been recoded by the two recoders 310a and 310b.

[0061] FIG. 6B illustrates an example recoding operation wherein one expand instruction I_0, one expandable instruction I_1, and two normal (e.g., non-expand and non-expandable) instructions I_2 and I_3 are fetched from an instruction cache during a clock cycle-0 of computer system 100. Instruction I_0 is dispatched to a recoder 310a. Instruction I_1 is dispatched to a recoder 310b. Because instruction

I_0 is an expand instruction, recoder 310a obtains information needed to recode expandable instruction I_1 and passes this information to recoder 310b. Recoder 310b then uses the passed information from recoder 310a to recode the expandable instruction I_1. As shown in FIG. 6B, recoders 310a and 310b operate together to recode instructions I_0 and I_1 and form a single recoded instruction during one clock cycle of computer system 100. In the next clock cycle of computer system 100, instruction I_2 is dispatched to recoder 310a and instruction I_3 is dispatched to recoder 310b. This time, because both of these instructions are normal instructions, recoders 310a and 310b are able to operate independently and recode instructions I_2 and I_3 during a single clock cycle of computer system 100. At the end of two clock cycles of computer system 100, all four instructions I_0, I_1, I_2, and I_3 have been recoded by the two recoders 310a and 310b to form three recoded instructions.

[0062] FIG. 6C illustrates an example recoding operation wherein one expand instruction I_1, one expandable instruction I_2, and two normal (e.g., non-expand and non-expandable) instructions I_0 and I_3 are fetched from an instruction cache during a clock cycle-0 of computer system 100. Instruction I_0 is dispatched to a recoder 310a. Instruction I_1 is dispatched to a recoder 310b. Because instruction I_0 is a normal instruction, recoder 310a is able to recode the instruction without any input from another recoder. Because instruction I_1 is an expand instruction, recoder 310b obtains information needed to recode expandable instruction I_2 and passes this information to recoder 310b via information storage buffer 312. Buffer 312 stores the information needed to recode instruction I_2 until instruction I_2 can be dispatched to recoder 310a. Recoder 310a then uses the passed information from recoder 310b to recode the expandable instruction I_2 during a subsequent clock cycle (clock cycle 2) of computer system 100. Because instruction I_3 is a normal instruction, recoder 310b is able to recode the instruction without any input from another recoder. Again, at the end of two clock cycles of computer system 100, all four instructions I_0, I_1, I_2, and I_3 have been recoded by the two recoders 310a and 310b to form three recoded instructions.

[0063] FIG. 6D illustrates an example recoding operation wherein one expand instruction I_2, one expandable instruction I_3, and two normal (e.g., non-expand and non-expandable) instructions I_0 and I_1 are fetched from an instruction cache during a clock cycle-0 of computer system 100. Instruction I_0 is dispatched to a recoder 310a. Instruction I_1 is dispatched to a recoder 310b. Because both of these instructions are normal instructions, recoders 310a and 310b are able to operate independently and recode instructions I_0 and I_1 during a single clock cycle of computer system 100. Because instruction I_2 is an expand instruction, recoder 310a obtains information needed to recode expandable instruction I_3 and passes this information to recoder 310b. Recoder 310b then uses the passed information from recoder 310a to recode the expandable instruction I_3. As shown in FIG. 6B, recoders 310a and 310b operate together to recode instructions I_2 and I_3 and form a single recoded instruction during one clock cycle of computer system 100. At the end of two clock cycles of computer system 100, all four instructions I_0, I_1, I_2, and I_3 have been recoded by the two recoders 310a and 310b to form three recoded instructions.

[0064] FIG. 6E illustrates an example recoding operation wherein one expand instruction I_3 and three normal (e.g., non-expand and non-expandable) instructions I_0, I_1 and I_2 are fetched from an instruction cache during a clock cycle-0 of computer system 100. Instruction I_0 is dispatched to a recoder 310a. Instruction I_1 is dispatched to a recoder 310b. Because both of these instructions are normal instructions, recoders 310a and 310b are able to operate independently and recode instructions I_0 and I_1 during a single clock cycle of computer system 100. Instruction I_2 is also dispatched to recoder 310a. Because instruction I_2 is a normal instruction, recoder 310a is able to recode the instruction without any input from another recoder. Instruction I_3 is dispatched to recoder 310b. Because instruction I_3 is an expand instruction, recoder 310b obtains information needed to recode expandable instruction I_4 and passes this information to recoder 310a via information storage buffer 312. Recoder 310a then uses the passed information from recoder 310b to recode the expandable instruction I_4.

[0065] FIG. 6F illustrates an example recoding operation wherein one incorrect (i.e., undesired) instruction I_0, one expand instruction I_1, one expandable instruction I_2, and one normal instruction I_3 are fetched from an instruction cache during a clock cycle-0 of computer system 100. Instruction I_0 is not recoded because it is an incorrect instruction. Instruction I_1 is dispatched to a recoder 310b. Because instruction I_1 is an expand instruction, recoder 310b obtains information needed to recode expandable instruction I_2 and passes this information to recoder 310a via information storage buffer 312. Recoder 310a then uses the passed information from recoder 310b to recode the expandable instruction I_2. Because instruction I_3 is a normal instruction, recoder 310b is able to operate independently and recode instruction I_3 without any input from another recoder. As shown in FIG. 6F, at the end of two clock cycles of computer system 100, the three instructions I_1, I_2, and I_3 have been recoded by the two recoders 310a and 310b to form two recoded instructions.

[0066] As already noted, alternate embodiments of the invention may have more than two recoders. These embodiments would operate similarly to the two recoder embodiment described above. How such embodiments are implemented would be apparent to persons skilled in the relevant computer arts given the description of the invention herein.

Conclusions

[0067] While various embodiments of the present invention have been described above, it should be understood that they have been presented by way of example, and not limitation. It will be apparent to persons skilled in the relevant computer arts that various changes in form and detail can be made therein without departing from the spirit and scope of the invention.

[0068] For example, in addition to recoding system implementations using hardware (e.g., within or coupled to a Central Processing Unit ("CPU"), microprocessor, microcontroller, digital signal processor, processor core, System

on Chip ("SOC"), or any other programmable device), implementations may also be embodied in software (e.g., computer readable code, program code, instructions and/or data disposed in any form, such as source, object or machine language) disposed, for example, in a computer usable (e.g., readable) medium configured to store the software. Such software enables the function, fabrication, modeling, simulation, description and/or testing of the apparatus and methods described herein. For example, this can be accomplished through the use of general programming languages (e.g., C, C++), GDSII databases, hardware description languages (HDL) including Verilog HDL, VHDL, AHDL (Altera HDL) and so on, or other available programs, databases, and/or circuit (i.e., schematic) capture tools. Such software can be disposed in any known computer usable medium including semiconductor, magnetic disk, optical disc (e.g., CD-ROM, DVD-ROM, etc.) and as a computer data signal embodied in a computer usable (e.g., readable) transmission medium (e.g., carrier wave or any other medium including digital, optical, or analog-based medium). As such, the software can be transmitted over communication networks including the Internet and intranets.

[0069] It is understood that the apparatus and methods described herein may be included in a semiconductor intellectual property core, such as a microprocessor core (e.g., embodied in HDL) and transformed to hardware in the production of integrated circuits. Additionally, the apparatus and methods described herein may be embodied as a combination of hardware and software. Thus, the present invention should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.